SoC Linux 道場【其ノ五】

Linux カーネルの入手とコンパイル、カスタム・ドライバの作成とコンパイル（その 1）

Ver.13.1

2017 年 8 月 Rev.2

目次

1. はじめに

今回は、アルテラ SoC FPGA 向け Yocto ソース・パッケージを利用した Linux® カーネルの入手とコンパイルの手順について解説します。

RocketBoards.org ポータル・サイトから Linux カーネルの git ツリーのクローンを作成し、その後、カーネル・コンフィギュレーションとコンパイルを実行します。

生成物として zImage という Linux カーネルのバイナリ・イメージができます。

この zImage ファイルを本連載の【其ノ壱】“ビルド済み SD カード・イメージを使用した Helio ボードでの Linux ブートの確認”で作成した microSD カード内にある zImage ファイルと差し換えることで、今回コンパイルした Linux カーネルの起動を Helio ボードで確認することができます。

また、「カスタム・ドライバの作成とコンパイル（その 1）」として、簡単な Hello ドライバ（Hello メッセージを表示するドライバ）をコンパイルして Helio ボードに NFS サーバー経由で転送してから実行してみます。

尚、この資料の説明で使用している主な開発環境は以下の通りです。

【表 1.1】この資料の説明で使用している主な開発環境

項番	項目	内容
1	ホスト OS	Microsoft Windows 7 Professional sp1 日本語版（64 bit）
2	ゲスト OS	Vine Linux 6.5 x86_64 この資料では、Linux 開発環境として、Vine Linux ディストリビューションを使用します。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
3	仮想 OS 実行環境	OS を仮想的に実行するための環境です。 この説明では、「VMware Player for Windows」と呼ばれるフリーウェア・ソフトを使用しています。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
4	クロス・コンパイラ	ターゲット（Helio）向けの実行イメージを生成するためのコンパイラです。 この説明では、Linaro から提供されている 32-bit ARMv7 Cortex-A 向け Linux GNU クロス・ツールチェーンを使用しています。 詳細については、SoC Linux 道場【其ノ参】を参照ください。
5	Linux カーネル ソース・コード	アルテラ SoC FPGA 向け Linux カーネル・ソース・コード RocketBoards.org ポータル・サイトの以下のページに行くと、アルテラ SoC FPGA 向けの各バージョン／リリースの Linux カーネル・ソース・コードがあります。 この説明では、 rel_socfpga-3.9-rel_14.02.02 というバージョン／リリースを使用しています。 ■ アルテラ SoC FPGA 向けの各バージョン／リリースの Linux カーネルのページ https://github.com/altera-opensource/linux-socfpga ■ 詳細は、以下の「Linux - Getting Started Using Git Trees」を参照してください。 http://www.rocketboards.org/foswiki/Documentation/GitGettingSTarted

6	Helio ボード	動作確認でターゲット・ボードとして使用する、アルテラ Cyclone® V SoC を搭載した マクニカ Helio ボードです。 Helio には複数のリビジョンが存在しますが、この資料では、Rev1.2 または Rev1.3 を使用して動作確認を行っています。 ■ Helio ボード Rev1.2 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev12 ■ Helio ボード Rev1.3 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev13

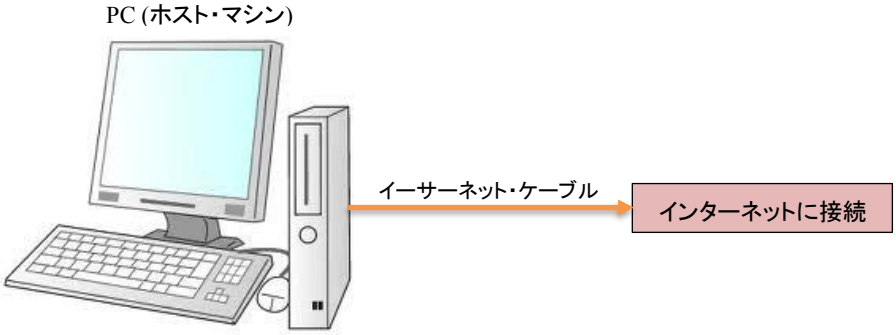
2. Linux カーネルの入手とコンパイル

本章はご自分の環境で、Helio（アルテラ SoC FPGA）用のカーネルを入手する際の参考にしてください。

2-1. インターネットに接続するためのネットワーク設定の変更

この連載において、ここまでのネットワーク設定と接続としては、ホスト PC と Helio を 1 対 1 で接続する閉じた環境を構成していました。

git ツリーのクローンを作成し、Linux カーネルをコンパイルする場合、外部のインターネットと接続するので、ネットワーク設定を変更する必要があります。



【図 2-1.1】ホスト PC をインターネットに接続する

会社内の LAN をご使用の場合は、プロキシの設定などがある場合もありますので、社内のネットワーク管理者に確認してください。

ここでの説明は、基本的な設定として参考にしてください。

【注記】

SoC Linux 道場【其ノ弐】で、Helio の IP アドレスを付与するために、Vine Linux 上で「端末」を開いて shell が起動すると、DHCP サーバーが起動するように設定しました。

今回の【其ノ五】では、外部のインターネットに接続するため、DHCP サーバーが動作しているとネットワーク上問題が発生することも考えられます。

【其ノ弐】で、DHCP サーバーが起動するように設定している場合は、外部のインターネットに接続する前に、DHCP サーバーを停止させてください。

DHCP サーバーを停止させる方法としては、Vine Linux 上で「端末」を開いて shell が起動したら、以下のコマンドを入力します。

```
[tori@Vine65 ~]$ su -
パスワード:
[root@Vine65 ~]# /etc/init.d/dhcpd stop
dhcpd を停止中:                                            [  OK  ]
[root@Vine65 ~]# su - tori
```

※ 新たに「端末」を開いて shell が起動するたびに、DHCP サーバーが起動するので、その都度 DHCP サーバーを停止させてください。

2-1-1. Windows のネットワーク設定

(1) Helio とホスト PC がネットワーク・ケーブルで接続されている場合はケーブルを外します。

(2) Windows のスタート・メニューから「コントロール パネル」を開きます。

(3) コントロール・パネルの項目から、「ネットワークと共有センター」をクリックします。

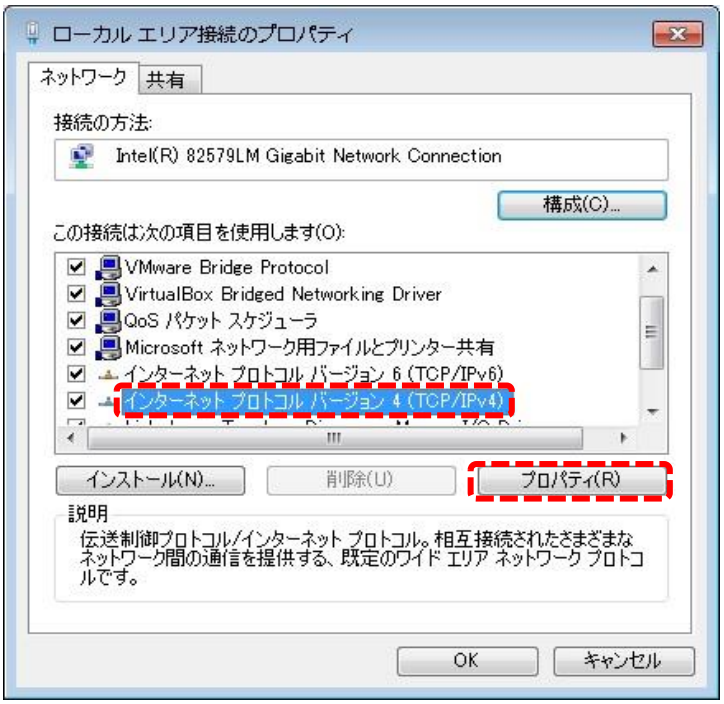
(4) 次の図のように、「アダプタの設定の変更」をクリックします。

【図 2-1-1.1】アダプタの設定の変更

(5) 「ローカル エリア接続」があるはずですので、これを右クリックして「プロパティ」を選択します。

【図 2-1-1.2】「ローカル エリア接続」を右クリックして「プロパティ」を選択

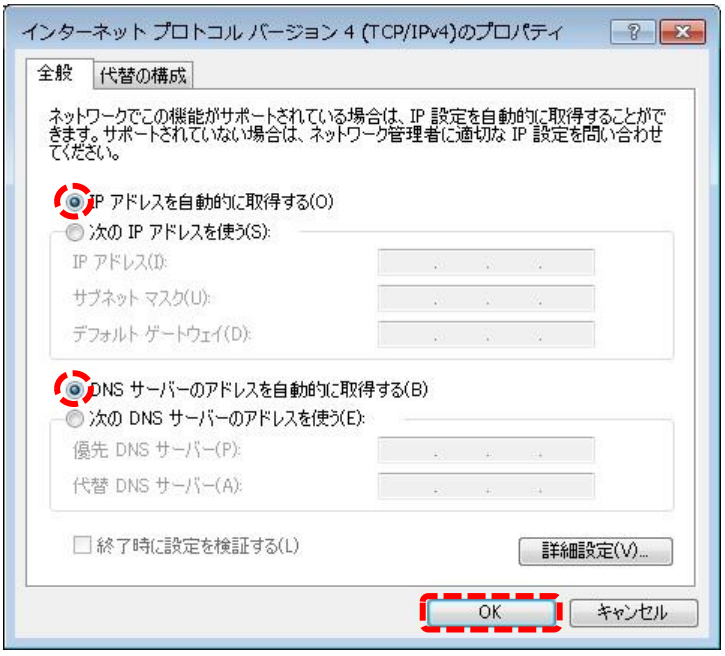
(6) 「インターネット プロトコル バージョン 4 (TCP/IPv4)」をハイライトして、「プロパティ」をクリックします。



【図 2-1-1.3】「インターネット プロトコル バージョン 4 (TCP/IPv4)」のプロパティを開く

(7) 通常ご使用のインターネット環境の設定に合わせて、「IP アドレス」と「サブネット マスク」を設定します。
この例では、「IP アドレスを自動的に取得する」、「DSN サーバーのアドレスを自動的に取得する」に設定しています。固定アドレスをお使いの場合はアドレスを設定します。

設定したら、「OK」をクリックします。



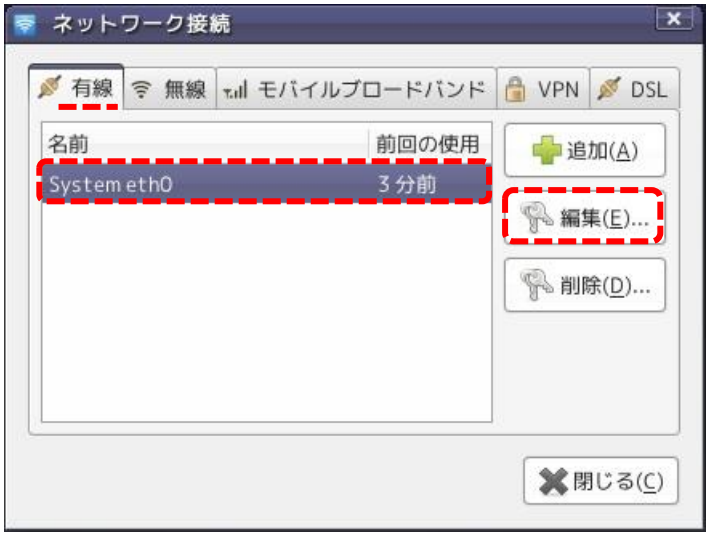
【図 2-1-1.4】「IP アドレス」と「サブネット マスク」の設定

2-1-2. Vine Linux のネットワーク設定

(1) Vine Linux のメニュー・バーから、「システム」⇒「設定」⇒「インターネットとネットワーク」⇒「ネットワーク接続」を選択します。

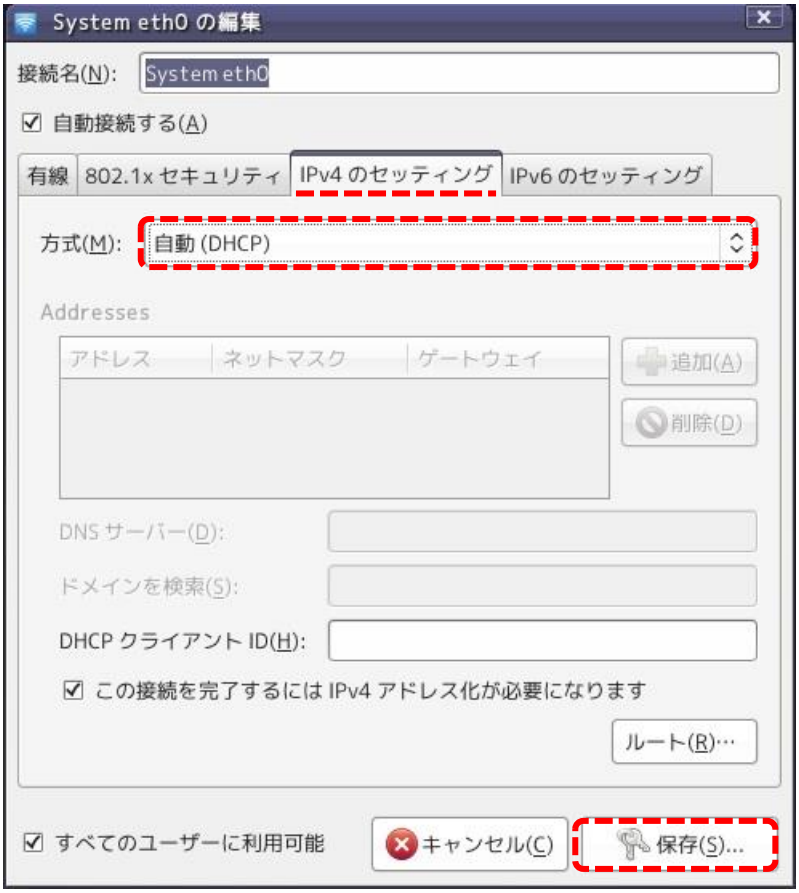
【図 2-1-2.1】「ネットワーク接続」を選択

(2) “ネットワーク接続”ウインドウが出たら、有線タブで‘System eth0’をハイライトして、「編集」をクリックします。



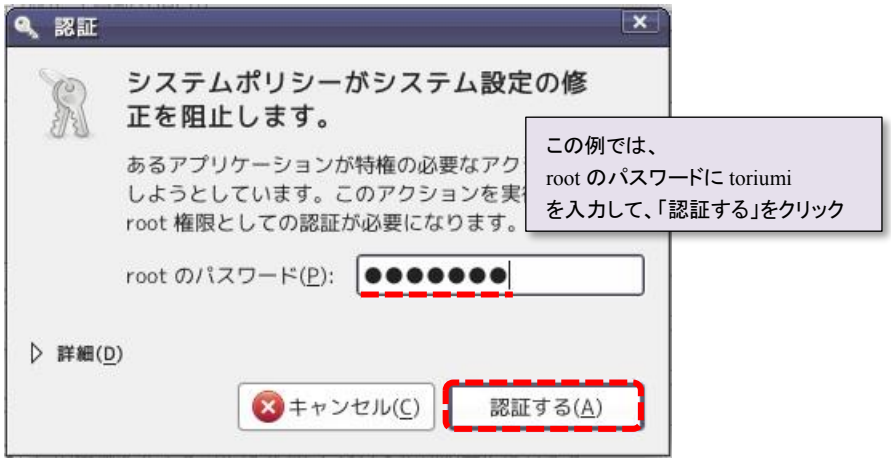
【図 2-1-2.2】有線タブで‘System eth0’をハイライトして、「編集」をクリック

(3) “System eth0 の編集”ウインドウが出たら、IPv4 のセッティング・タブの方式で‘自動（DHCP）’を選択して、「保存」をクリックします。



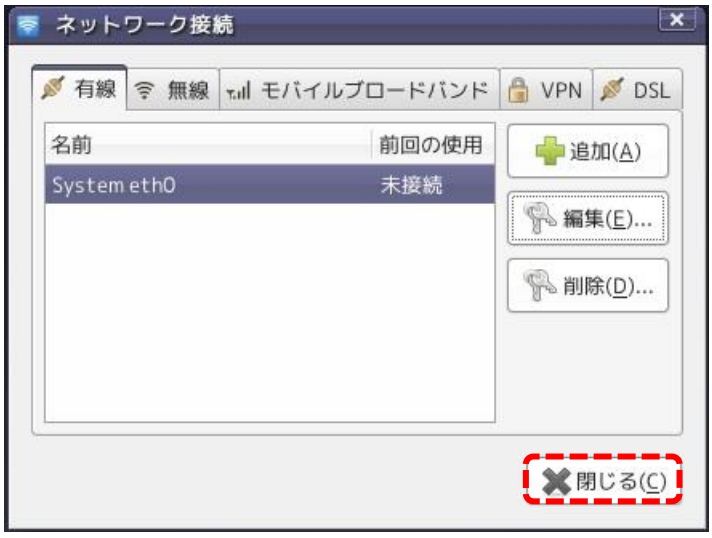
【図 2-1-2.3】IPv4 のセッティング・タブの方式で‘自動（DHCP）’を選択

(4) “認証”ウインドウが出たら、‘root のパスワード’（この例では、toriumi）を設定して、「認証する」をクリックします。



【図 2-1-2.4】システム設定の認証

(5) “ネットワーク接続”ウインドウで、「閉じる」をクリックします。

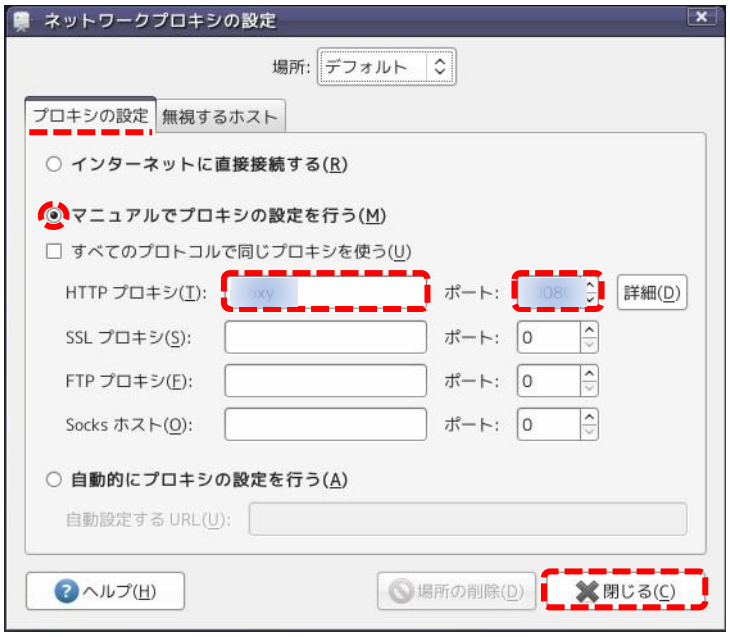


【図 2-1-2.5】“ネットワーク接続”ウインドウを閉じる

(6) 会社内の LAN などで、プロキシの設定があるような場合は、Vine Linux のメニュー・バーから、「システム」⇒「設定」⇒「インターネットとネットワーク」⇒「ネットワーク プロキシ」を選択します。

【図 2-1-2.6】「ネットワーク プロキシ」を選択

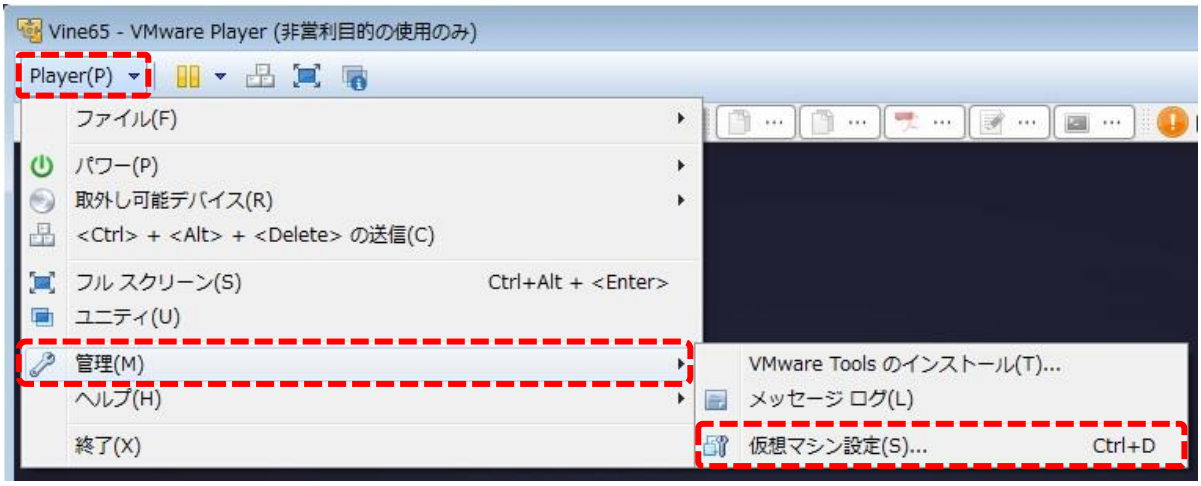
(7) “ネットワーク プロキシ” ウインドウが出たら、プロキシの設定タブで ‘マニュアルでプロキシの設定を行う’ を選択して、Windows と同じ HTTP プロキシとポートを設定します。設定が終わったら、「閉じる」をクリックします。プロキシ設定で不明な点は社内のネットワーク管理者に確認してください。



【図 2-1-2.7】“ネットワーク プロキシ” ウインドウでの設定

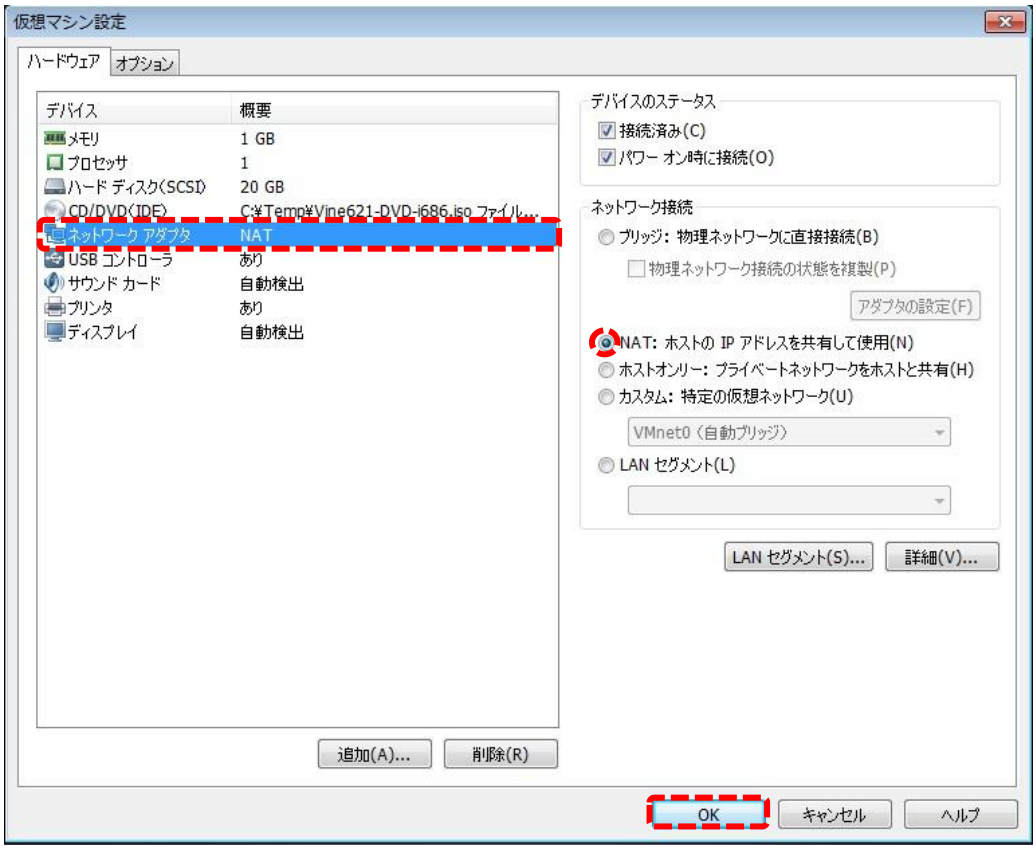
2-1-3. VMware のネットワーク設定

(1) VMware の“Player”をクリックし、「管理」⇒「仮想マシン設定」を選択します。



【図 2-1-3.1】VMware の“Player”をクリックし、「管理」⇒「仮想マシン設定」を選択

(2) “仮想マシン設定”ウインドウが出たら、ハードウェア・タブで‘ネットワーク アダプタ’をハイライトして、ネットワーク接続で「NAT」を選択します。設定が終わったら、「OK」をクリックします。



【図 2-1-3.2】“仮想マシン設定”ウインドウで「NAT」を選択

2-1-4. インターネットへの接続確認

(1) ネットワーク設定を変更したら、ホスト PC に外部のインターネットとネットワーク・ケーブルで接続します。

(2) Vine Linux 上で Firefox ブラウザを起動します。Vine Linux のメニュー・バーから、「アプリケーション」⇒「インターネット」⇒「Fx ウェブブラウザ（firefox）」を選択（または、アイコンをクリック）します。

【図 2-1-4.1】Vine Linux 上で Firefox ブラウザを起動

(3) Firefox ブラウザが起動して、インターネットに接続できれば設定成功です。

【図 2-1-4.2】インターネットに接続できれば成功

2-2. Linux カーネルの入手

Linux 上の一般ユーザ（この例では tori）のホーム・ディレクトリ（/home/tori）から以下のコマンドを実行します。

(1) まずカーネルを入手する前に git ツリーを使ってソース・コードをビルドすることができるように、次のコマンドを実行して、Vine Linux に以下の必要なプログラムをインストールしておきます。

```
[tori@Vine65 ~]$ cd ~
[tori@Vine65 ~]$ sudo apt-get update
[tori@Vine65 ~]$ sudo apt-get dist-upgrade
[root@Vine65 ~]$ sudo apt-get install git-core texi2html chrpath
```

【注記】

社内 LAN などでプロキシがある場合は、上記の apt-get コマンドを以下のように置き換えて実行してください（詳しくはネットワーク管理者に確認してください）。

なお、以下のコマンド列において、**proxy_server** と **xxxxx** の表記部分は、実際にご使用のネットワーク環境に応じて置き換えてください。

proxy_server → プロキシ・サーバー名
xxxxx → ポート番号

```
[tori@Vine65 ~]$ sudo http_proxy=http://proxy_server:xxxxx apt-get update
[tori@Vine65 ~]$ sudo http_proxy=http://proxy_server:xxxxx apt-get install git-core texi2html chrpath
```

(2) GitHub の altera-opensource サイトから linux-socfpga カーネルの git ツリーのクローンを作成します（数十分かかります）。

作成が完了すると、“linux-socfpga”というディレクトリができます。

```
[tori@Vine65 ~]$ git clone https://github.com/altera-opensource/linux-socfpga
```

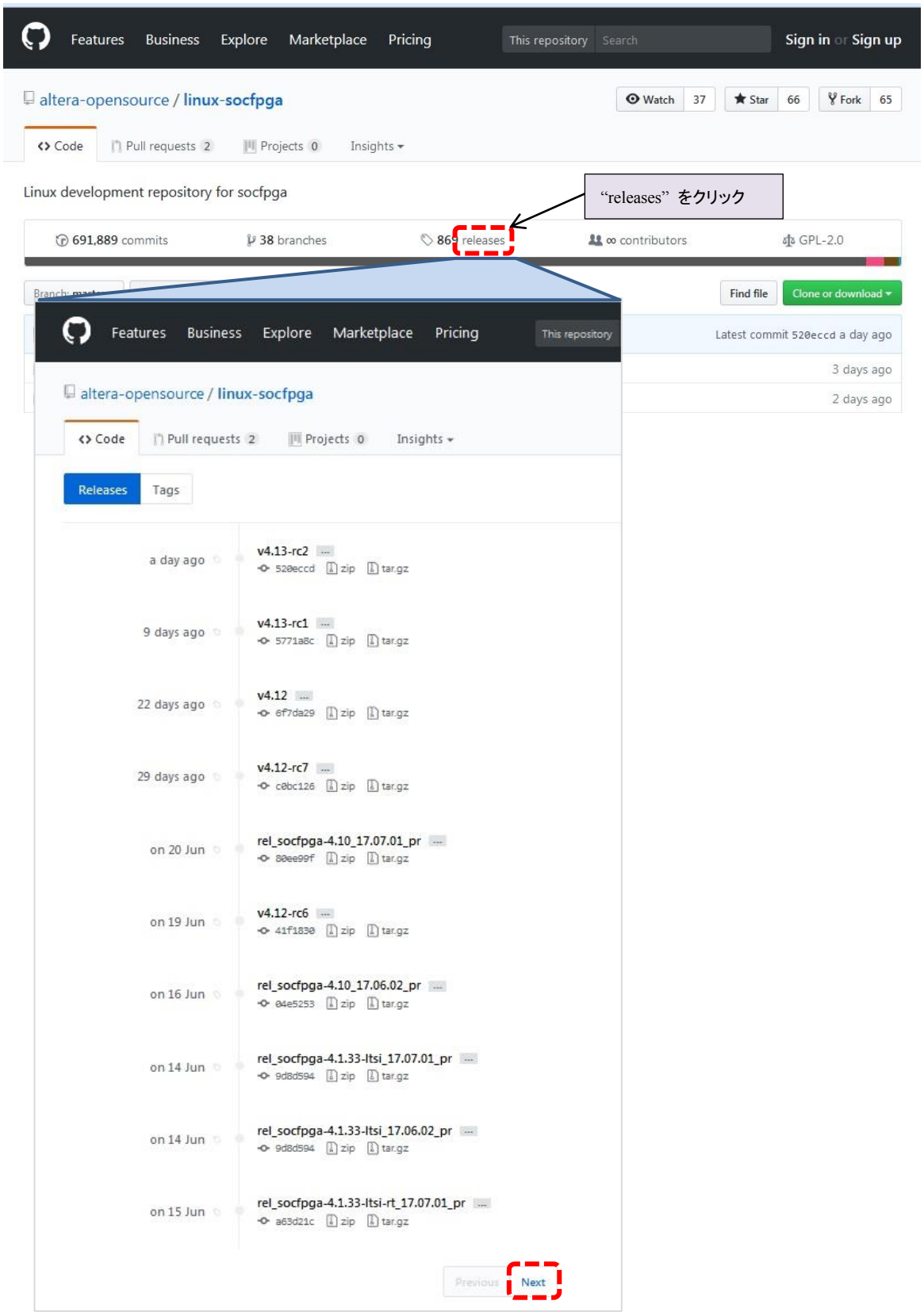
【注記】

```
error: unable to open object pack directory: /home/tori/linux-socfpga/.git/objects/pack: Too many open files
fatal: failed to read object 579a4bd2e2c0160f34d1ce8edf2a21fe12c063d1: Too many open files
```

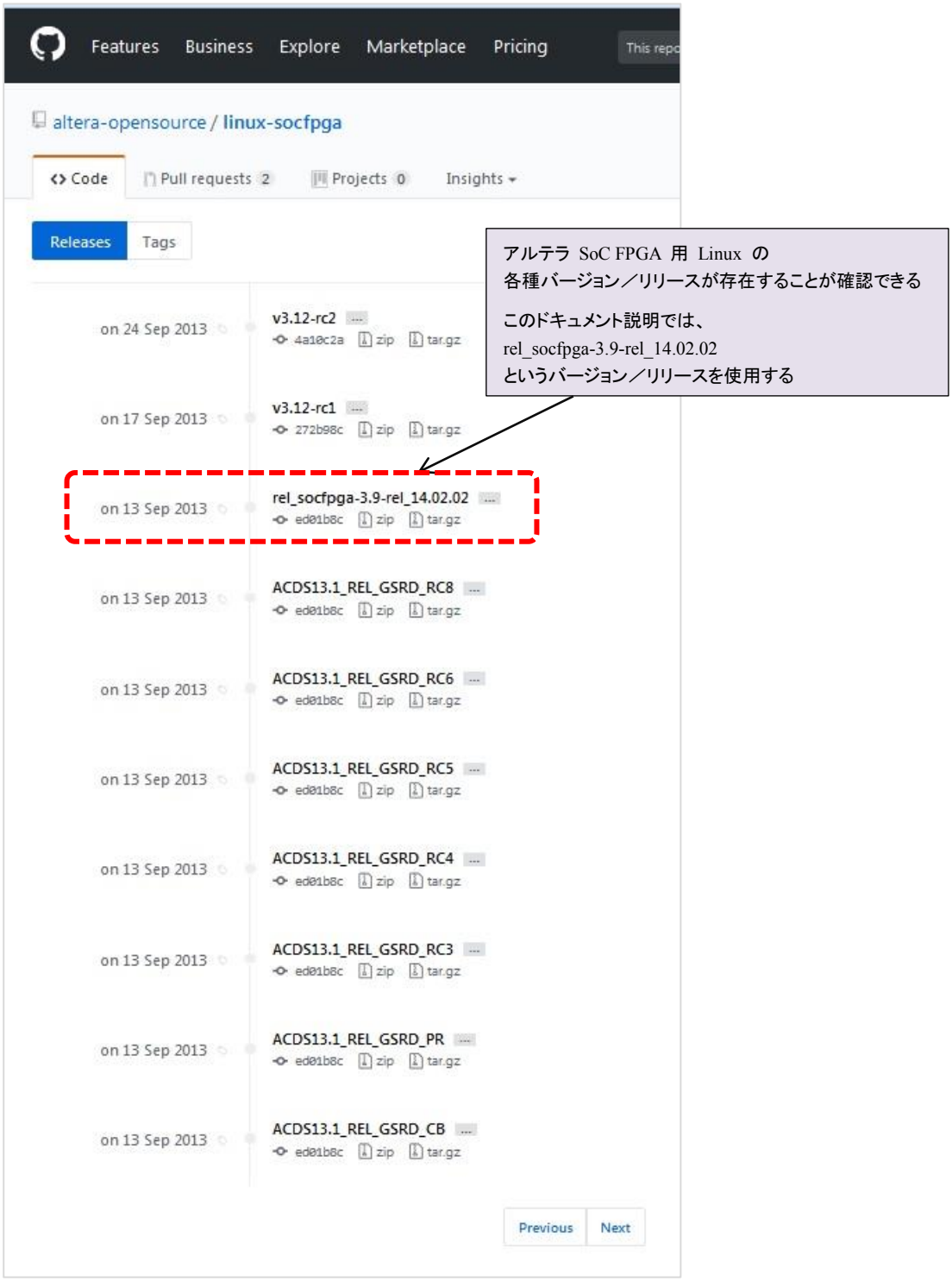
のエラーが出る場合は、以下のコマンドを実行してオープンできるファイル数の制限を増やしてから、再度 git clone コマンドを試してみてください。

```
[tori@Vine65 ~]$ ulimit -n
1024
[tori@Vine65 ~]$ su -
パスワード:
[root@Vine65 ~]# ulimit -n 65536
[root@Vine65 ~]# ulimit -n
65536
[root@Vine65 ~]# su - tori
[tori@Vine65 ~]$ ulimit -n
65536
```

(3) PC 上の Internet Explorer で、https://github.com/altera-opensource/linux-socfpga のアドレスを入力して、“releases” をクリックすると、アルテラ SoC FPGA 用 Linux の各種バージョン／リリースが存在することが確認できます（キャプチャ画面参照）。



【図 2-2.1】GitHub の altera-opensource サイトでの linux-socfpga カーネルのページ



【図 2-2.2】アルテラ SoC FPGA 用 Linux の各種バージョン／リリースの確認

(4) Linux カーネルの git ツリーのクローンが作成できたら、“linux-socfpga”ディレクトリに移動し、現在のソース・ツリーを元に新たなブランチを作成します。

コミット（この例では、rel_socfpga-3.9-rel_14.02.02 を使用）からブランチ（この例では、test_0 というブランチ名を使用）を作成して切り替えます。

```
[tori@Vine65 ~]$ cd linux-socfpga
[tori@Vine65 linux-socfpga]$ git checkout -b test_0 rel_socfpga-3.9-rel_14.02.02
```

2-3. Linux カーネルのコンパイル

次に入手した Linux カーネルをコンパイルします。

【注記】

SoC Linux 道場【其ノ参】で説明した、クロス・コンパイラがインストールされて、パスが設定されていて使用できることを前提としています。

make コマンドを実行してエラーが出るようであれば、**which arm-linux-gnueabihf-gcc** コマンドを実行して、クロス・コンパイラのパスが通っているか確認してください。
クロス・コンパイラのパスが正しく通っていない場合は、再度 SoC Linux 道場【其ノ参】を参照して必要な設定を行ってください。
または、次のコマンドを実行してクロス・コンパイラのパスを設定してください。

```
[tori@Vine65 linux-socfpga]$ export PATH=/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/:$PATH
```

(1) 以下のコマンドを実行して、【図 2-3.1】のような Linux カーネルのコンフィギュレーション・メニューを立ち上げます。

```
[tori@Vine65 linux-socfpga]$ make ARCH=arm socfpga_defconfig CROSS_COMPILE=arm-linux-gnueabihf-
[tori@Vine65 linux-socfpga]$ make ARCH=arm menuconfig CROSS_COMPILE=arm-linux-gnueabihf-
```

【注記】

make menuconfig を実行して、次のエラーが出て Linux カーネルのコンフィギュレーション画面が出ない場合は、下記 ①、② の手順を実行してから再度 make menuconfig を実行してください。

```
/usr/bin/ld: scripts/kconfig/mconf.o: シンボル 'stdscr' への未定義参照です
/lib64/libtinfo.so.5: error adding symbols: DSO missing from command line
collect2: エラー: ld はステータス 1 で終了しました
make[1]: *** [scripts/kconfig/mconf] エラー 1
make: *** [menuconfig] エラー 2
```

① linux-socfpga/scripts/kconfig/Makefile をエディタで開いて以下の記述を追加します。

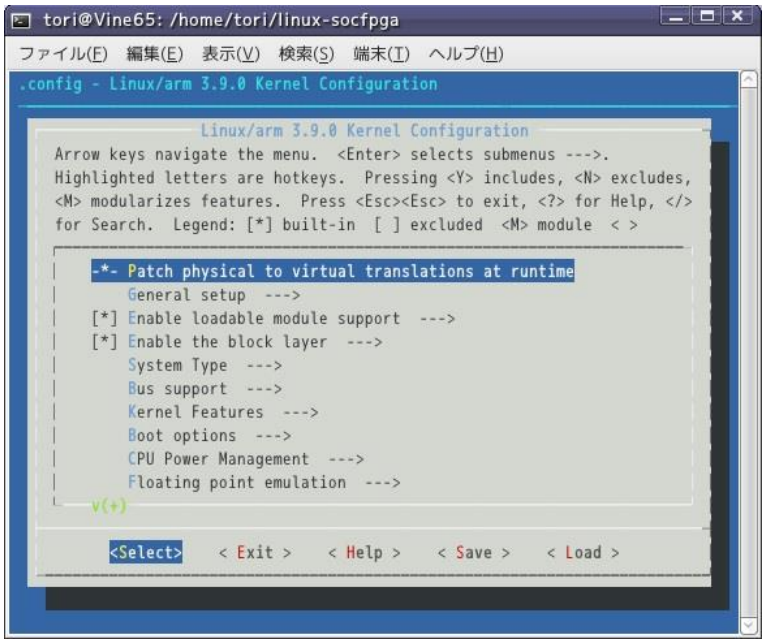
```
[tori@Vine65 linux-socfpga]$ leafpad scripts/kconfig/Makefile
```

```
HOST_EXTRACFLAGS += $(shell $(CONFIG_SHELL) $(check-lxdialog) -ccflags) ¥
                    -DLOCALE -I/usr/include/ncurses      <- -I/usr/include/ncurses を追加
HOST_LOADLIBES   += -ltinfo -lncurses -L/usr/lib64/      <- この行を新規追加
```

② linux-socfpga/scripts/kconfig/lxdialog/dialog.h ファイルをエディタで開き次のように変更します。

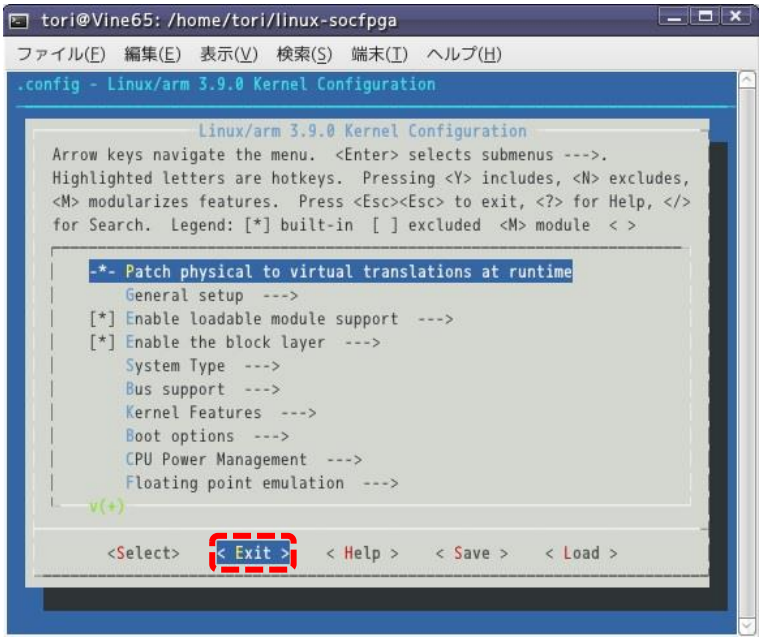
```
[tori@Vine65 linux-socfpga]$ leafpad scripts/kconfig/lxdialog/dialog.h
```

```
#include CURSES_LOC
          ↓
#include <ncurses.h>      #include CURSES_LOC を #include <ncurses.h> に変更
```



【図 2-3.1】カーネル・メニュー・ウィンドウ

(2) 必要であれば、カーネルのコンフィギュレーション・メニューで、矢印キー（→／←／↓／↑）を使用して操作し設定を行いますが、ここでは【図 2-3.2】のように単に「Exit」を選択し Enter キーを押してメニューを抜けます。



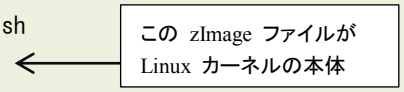
【図 2-3.2】「Exit」でカーネル・メニューを終了

(3) メニューを抜けたら以下のコマンドでカーネルをコンパイルします。お使いの PC のスペックや、ネットワーク環境にもよりますが、コンパイルには約 10～20 分程度かかります。

```
[tori@Vine65 linux-socfpga]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

(4) コンパイルが正常に終了すると、arch/arm/boot の下に zImage というファイルが生成されています。このファイルがカーネル本体になります。

```
[tori@Vine65 linux-socfpga]$ ls -l arch/arm/boot
合計 9792
-rwxr-xr-x 1 tori tori 6767640  7 月 25 11:56 Image*
-rw-r--r-- 1 tori tori    3179  7 月 25 08:47 Makefile
drwxr-xr-x 2 tori tori    4096  7 月 25 08:47 bootp/
drwxr-xr-x 2 tori tori    4096  7 月 25 11:56 compressed/
drwxr-xr-x 3 tori tori   69632  7 月 25 11:56 dts/
-rw-r--r-- 1 tori tori    1274  7 月 25 08:47 install.sh
-rwxr-xr-x 1 tori tori 3172920  7 月 25 11:56 zImage*
```



Helio 用の microSD カードを ホスト PC にマウントし、microSD カード内にある zImage を入れ換えれば、生成したカーネルで動作させることができます。

これは後程、試してみることにします。

2-4. Helio に接続するためのネットワーク設定の変更

ここで再び、ホスト PC と Helio を 1 対 1 で接続する閉じたネットワーク環境に設定を戻します。

2-4-1. Windows のネットワーク設定

(1) ホスト PC がネットワーク・ケーブルでインターネットに接続されている場合はケーブルを外します。

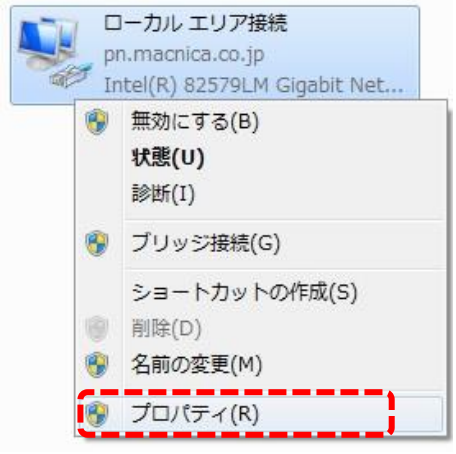
(2) Windows のスタート・メニューから「コントロール パネル」を開きます。

(3) コントロール・パネルの項目から、「ネットワークと共有センター」をクリックします。

(4) 次の図のように、「アダプタの設定の変更」をクリックします。

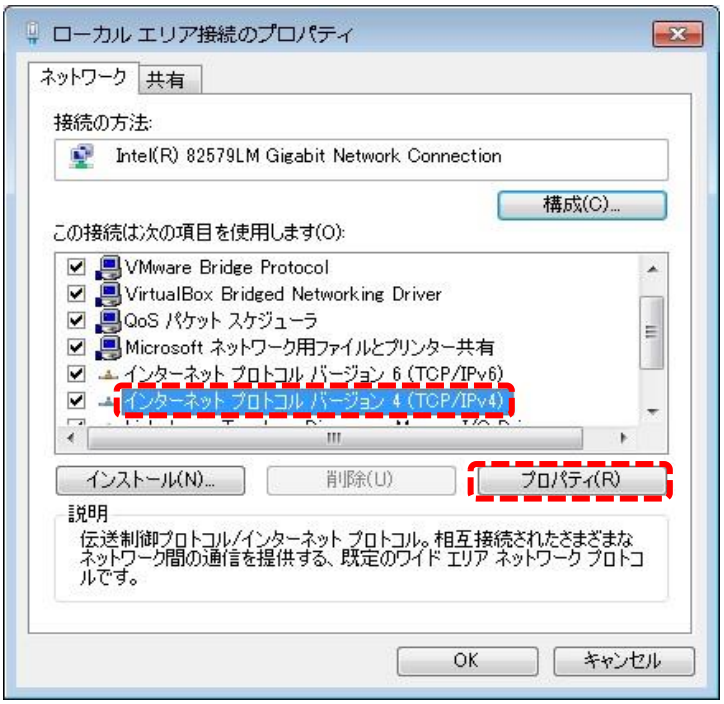
【図 2-4-1.1】アダプタの設定の変更

(5) 「ローカル エリア接続」があるはずですので、これを右クリックして「プロパティ」を選択します。



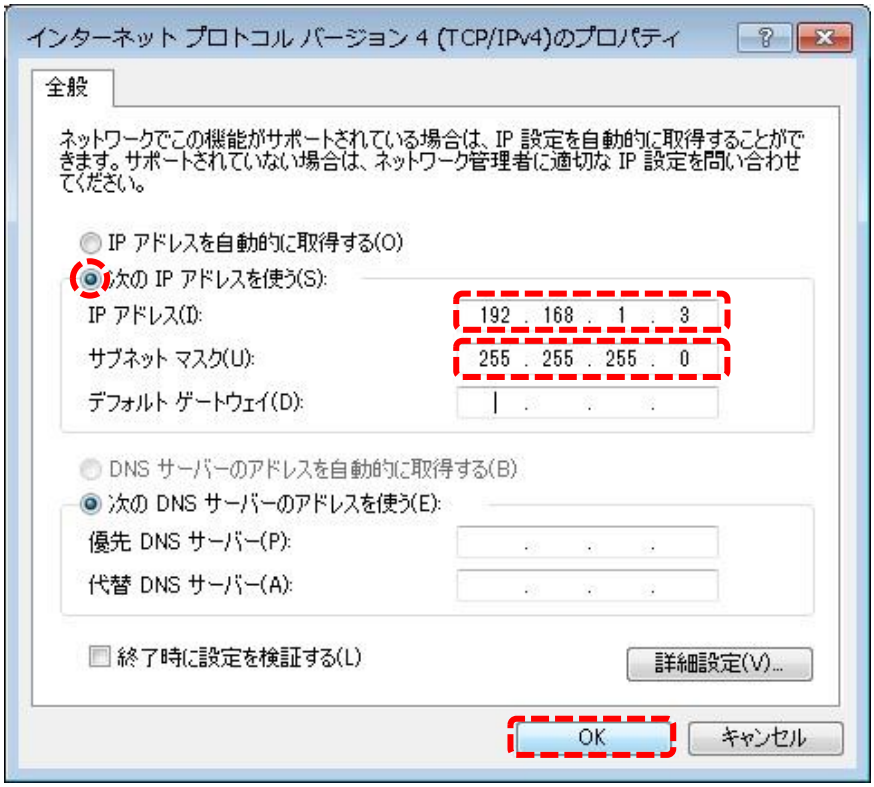
【図 2-4-1.2】「ローカル エリア接続」を右クリックして「プロパティ」を選択

(6)「インターネット プロトコル バージョン 4 (TCP/IPv4)」をハイライトして、「プロパティ」をクリックします。



【図 2-4-1.3】「インターネット プロトコル バージョン 4 (TCP/IPv4)」のプロパティを開く

(7)「次の IP アドレスを使う」を選択して、「IP アドレス」と「サブネット マスク」を設定します。
この例では、IP アドレスを 192.168.1.3、サブネット マスクを 255.255.255.0 に設定しています。
設定したら「OK」をクリックします。



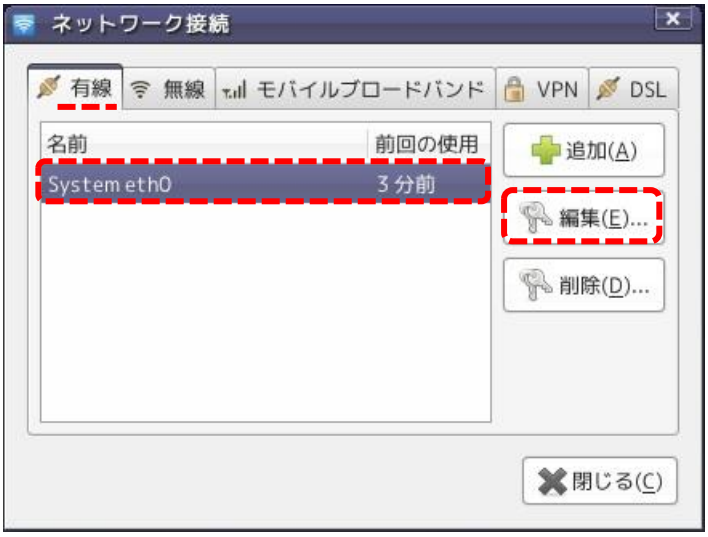
【図 2-4-1.4】「IP アドレス」と「サブネット マスク」の設定

2-4-2. Vine Linux のネットワーク設定

(1) Vine Linux のメニュー・バーから、「システム」⇒「設定」⇒「インターネットとネットワーク」⇒「ネットワーク接続」を選択します。

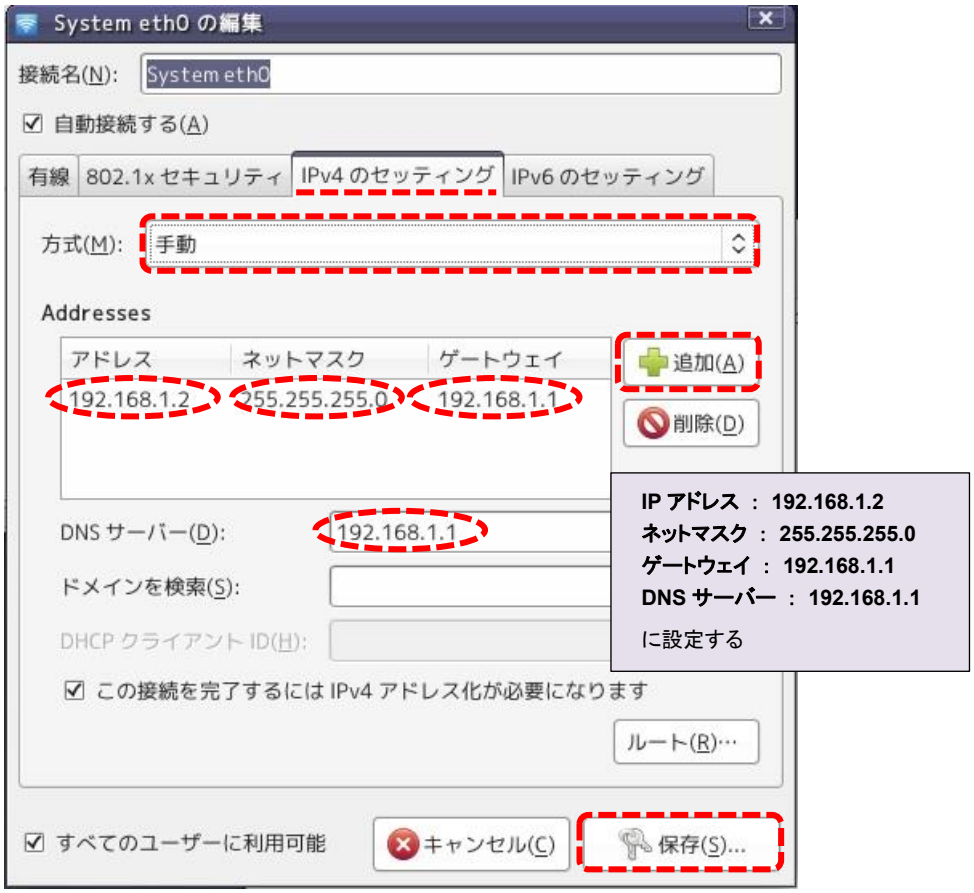
【図 2-4-2.1】「ネットワーク接続」を選択

(2) “ネットワーク接続”ウインドウが出たら、有線タブで‘System eth0’をハイライトして、「編集」をクリックします。



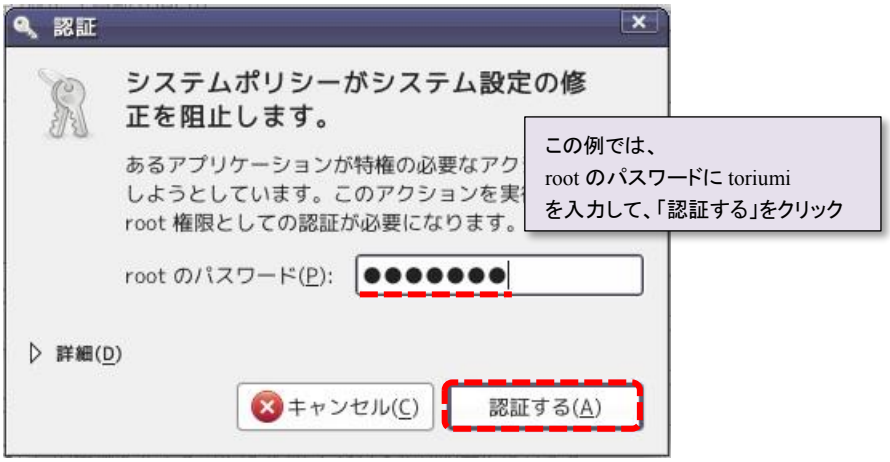
【図 2-4-2.2】有線タブで‘System eth0’をハイライトして、「編集」をクリック

(3) “System eth0 の編集”ウインドウが出たら、IPv4 のセッティング・タブの方式で‘手動’を選択し、「追加」をクリックして、IP アドレス、ネットマスク、ゲートウェイ、DNS サーバーを以下のように設定します。設定が終わったら「保存」をクリックします。



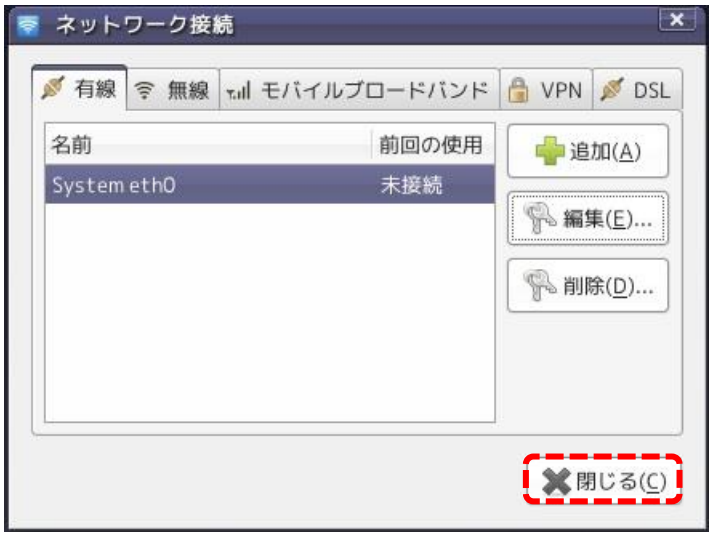
【図 2-4-2.3】IPv4 のセッティング・タブの設定内容

(4) “認証”ウインドウが出たら、‘root のパスワード’（この例では、toriumi）を設定して、「認証する」をクリックします。



【図 2-4-2.4】システム設定の認証

(5) “ネットワーク接続”ウインドウで、「閉じる」をクリックします。

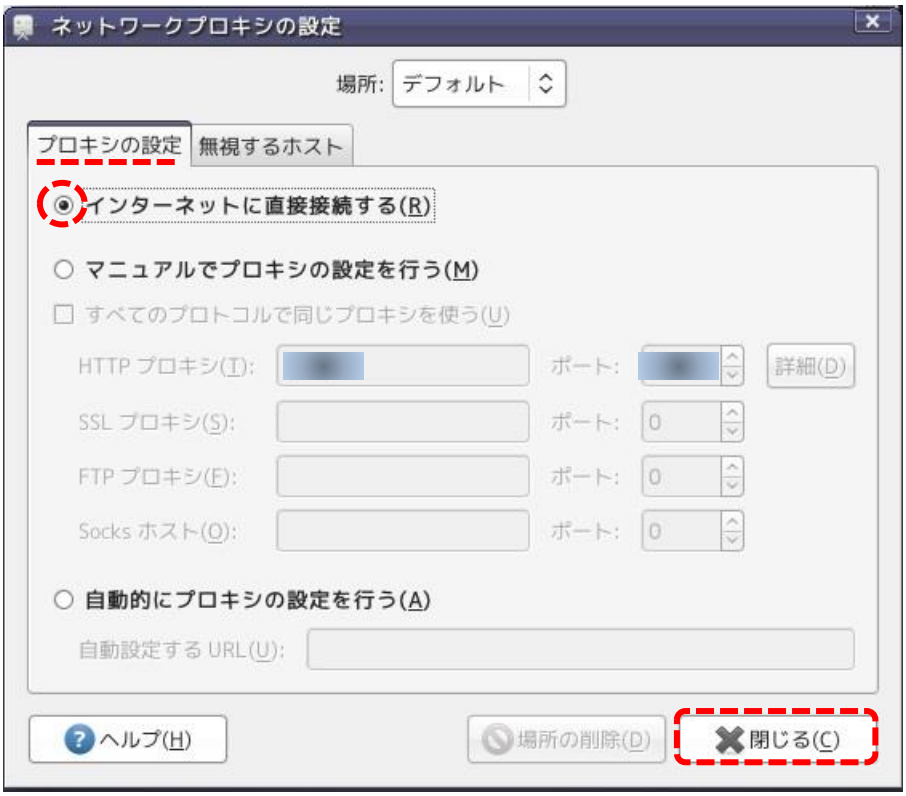


【図 2-4-2.5】“ネットワーク接続”ウインドウを閉じる

(6) Vine Linux のメニュー・バーから、「システム」⇒「設定」⇒「インターネットとネットワーク」⇒「ネットワーク プロキシ」を選択します。

【図 2-4-2.6】「ネットワーク プロキシ」を選択

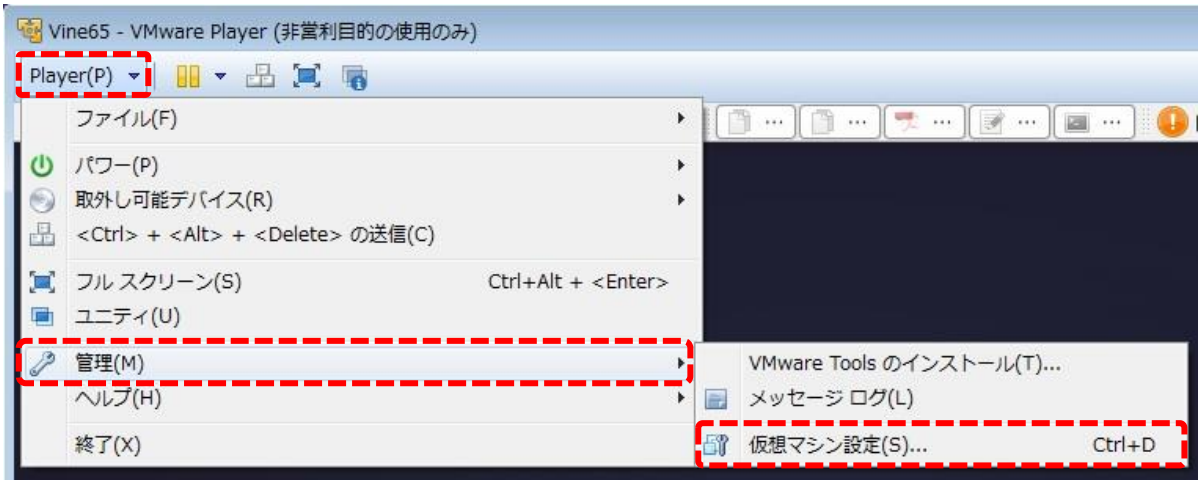
(7) “ネットワーク プロキシ” ウインドウが出たら、プロキシの設定タブで ‘インターネットに直接接続する’ を選択して、「閉じる」をクリックします。



【図 2-4-2.7】“ネットワーク プロキシ” ウインドウでの設定

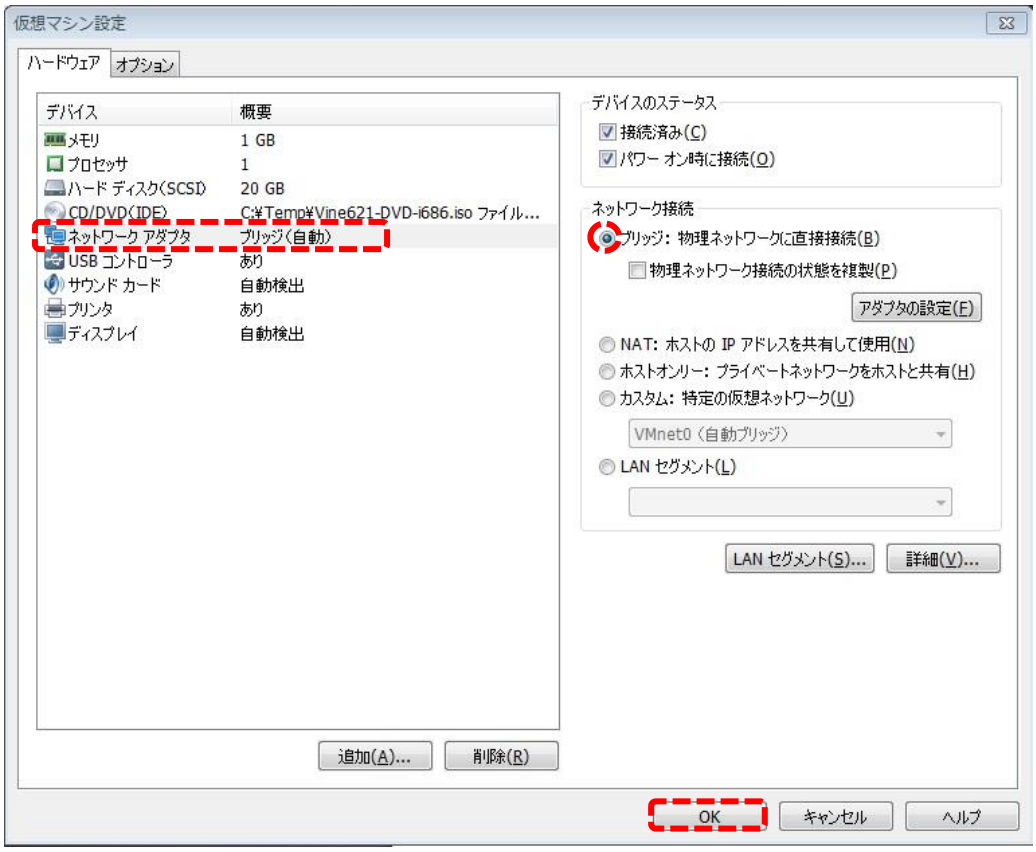
2-4-3. VMware のネットワーク設定

(1) VMware の“Player”をクリックし、「管理」⇒「仮想マシン設定」を選択します。



【図 2-4-3.1】VMware の“Player”をクリックし、「管理」⇒「仮想マシン設定」を選択

(2)“仮想マシン設定”ウインドウが出たら、ハードウェア・タブで‘ネットワーク アダプタ’をハイライトして、ネットワーク接続で「ブリッジ」を選択します。設定が終わったら、「OK」をクリックします。



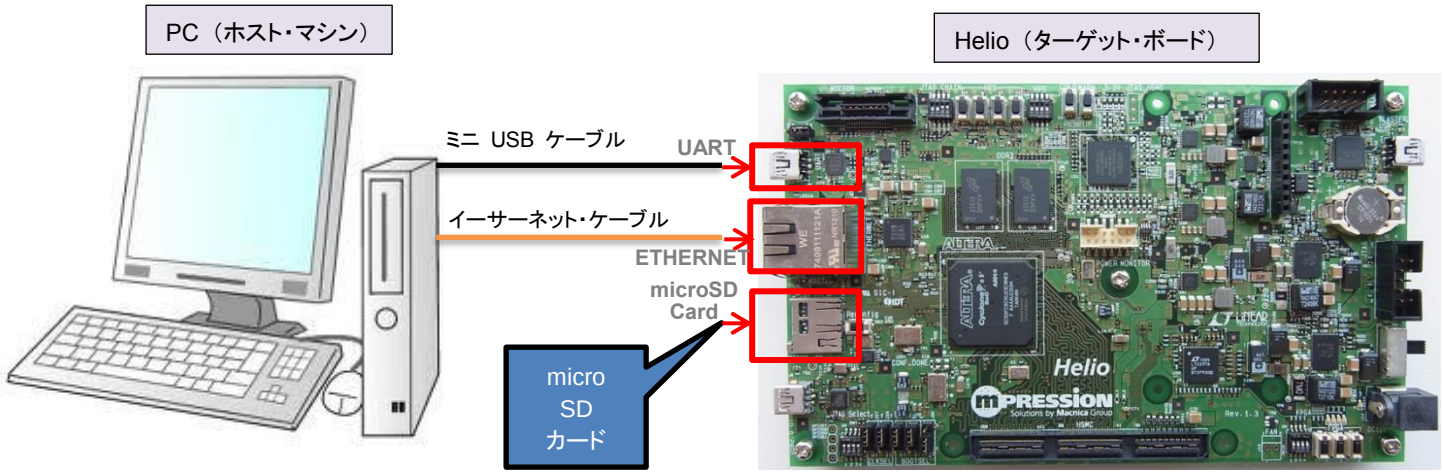
【図 2-4-3.2】“仮想マシン設定”ウインドウで「ブリッジ」を選択

2-4-4. Helio との接続確認

(1) ネットワーク設定を変更したら、PC と Helio を 1 対 1 でネットワーク接続して通信できるようにしますので、下図のように PC と Helio をイーサーネット・ケーブルで接続してください。

(2) また、USB-UART ケーブルも接続して、ホスト PC 上でターミナル・ソフト（TeraTerm など）を起動します。ボーレート「115200」、データ「8bit」、パリティ「none」、ストップビット「1bit」、フロー制御「none」に設定します。

(3) SoC Linux 道場【其ノ壱】で作成した microSD カードを Helio のカード・スロットに取り付けます。



【図 2-4-4.1】PC と Helio の接続図（全体）

(4) Helio の電源を入れて Linux を起動してみると、もともとの microSD カードでは、カーネルの起動メッセージとして以下のような内容になっているはずです。

```
Starting kernel ...

Booting Linux on physical CPU 0x0
Initializing cgroup subsys cpuset
Linux version 3.9.0 (jrucker@altera-linux-jr) (gcc version 4.7.3 20121106 (prerelease) (crosstool-NG
linaro-1.13.1-4.7-2012.11-20121123 - Linaro GCC 2012.11) ) #1 SMP Mon Nov 4 23:04:53 CST 2013
```

上記のログから、もともとの microSD カード内に入っている Linux カーネルのバージョンは 3.9.0 で、gcc のバージョンは 4.7.3 で、ツールチェーンは Linaro の GCC を使っていることがわかります。

2-5. コンパイルした Linux カーネルに差し換えて Helio を起動してみる

では、現在の microSD カード内の Linux のカーネル・イメージ・ファイル（zImage）を、今回コンパイルした zImage と差し換えて Helio を起動してみましょう。

2-5-1. Windows 側に zImage ファイルをコピーする

Samba サーバー経由で、Vine Linux 上にある今回コンパイルした zImage ファイルを、Windows 上にコピーします。このとき Helio の電源はまだ入れたまま（Linux が起動した状態）にしておいてください。

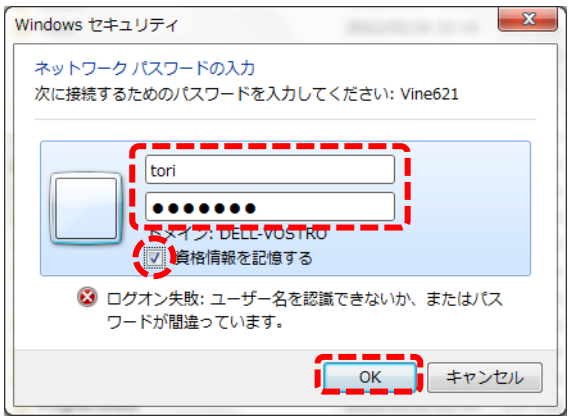
【注記】

以降の説明は、SoC Linux 道場【其ノ弐】で説明した、Linux マシンと Samba サーバーが既に用意・設定されていて、使用できることを前提としています。

(1) Windows 上のエクスプローラのアドレス・バーに以下のように「¥¥Vine65」または「¥¥192.168.1.2」と入力します。

【図 2-5-1.1】Windows のエクスプローラのアドレス・バーに Vine Linux のホスト名を入力

(2) ユーザ名とパスワードを入力するウィンドウが現れた場合は、smbpasswd で設定した ID（この例では tori）とパスワード（この例では toriumi）を入力します。



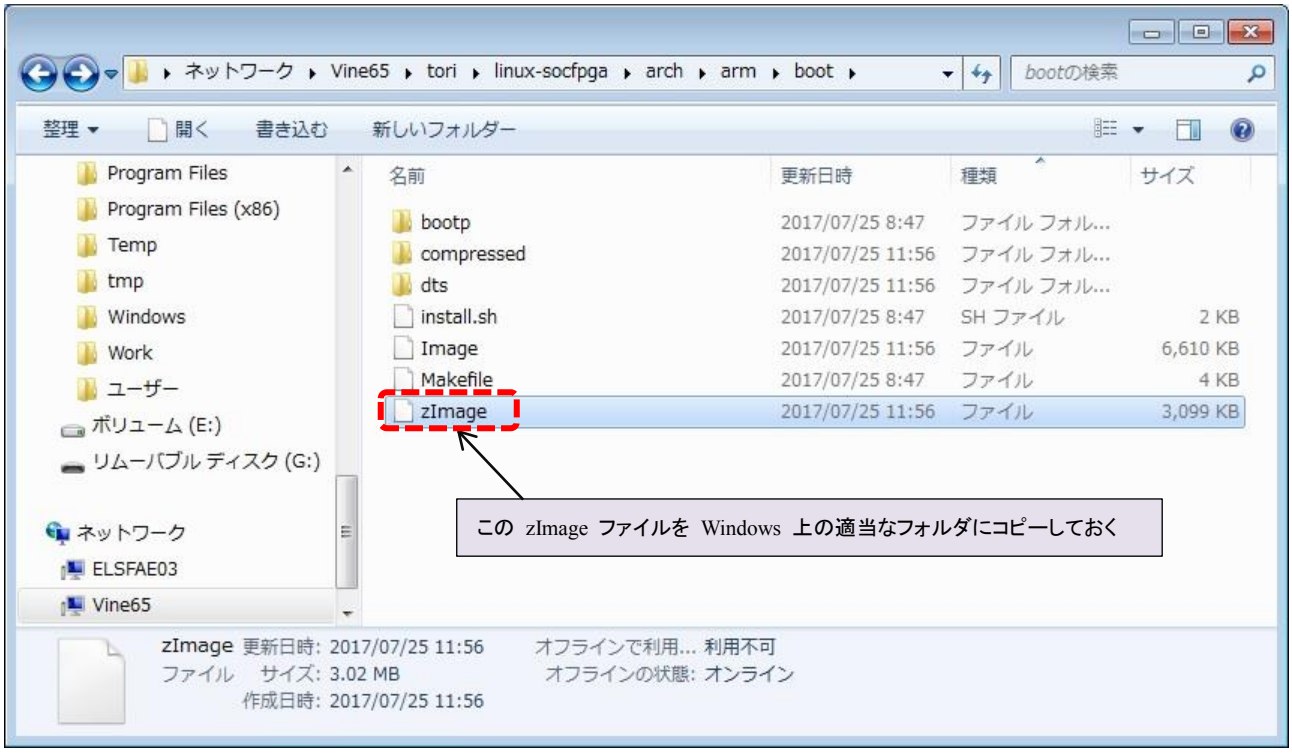
【図 2-5-1.2】Samba 用の ID とパスワードの入力

(3) 下図のように、Linux のファイル・システムがエクスプローラから見えれば成功です。

Samba サーバーがうまく起動すれば、エクスプローラで Linux と Windows の間で様々なファイルをやり取りすることができます。

【図 2-5-1.3】Windows のエクスプローラから Linux 側のファイルを確認

(4) Vine Linux 上のファイルが見えたら ¥tori¥linux-socfpga¥arch¥arm¥boot に行き、zImage ファイルを Windows 上の適当なフォルダ（Temp など）にコピーしておきます。



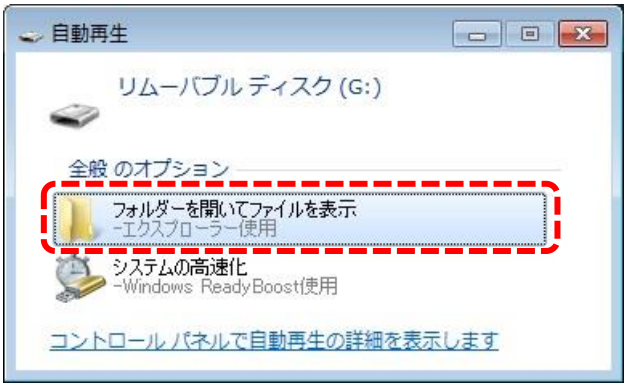
【図 2-5-1.4】¥tori¥linux-socfpga¥arch¥arm¥boot の下にある zImage ファイル

2-5-2. microSD カード内の zImage ファイルを差し換える

ホスト PC の SD カード・スロット（または USB カード・リーダを接続）に microSD カードを挿して、カード内の zImage と差し換えます。

(1) Helio の電源を落として、microSD カードを取り出します。

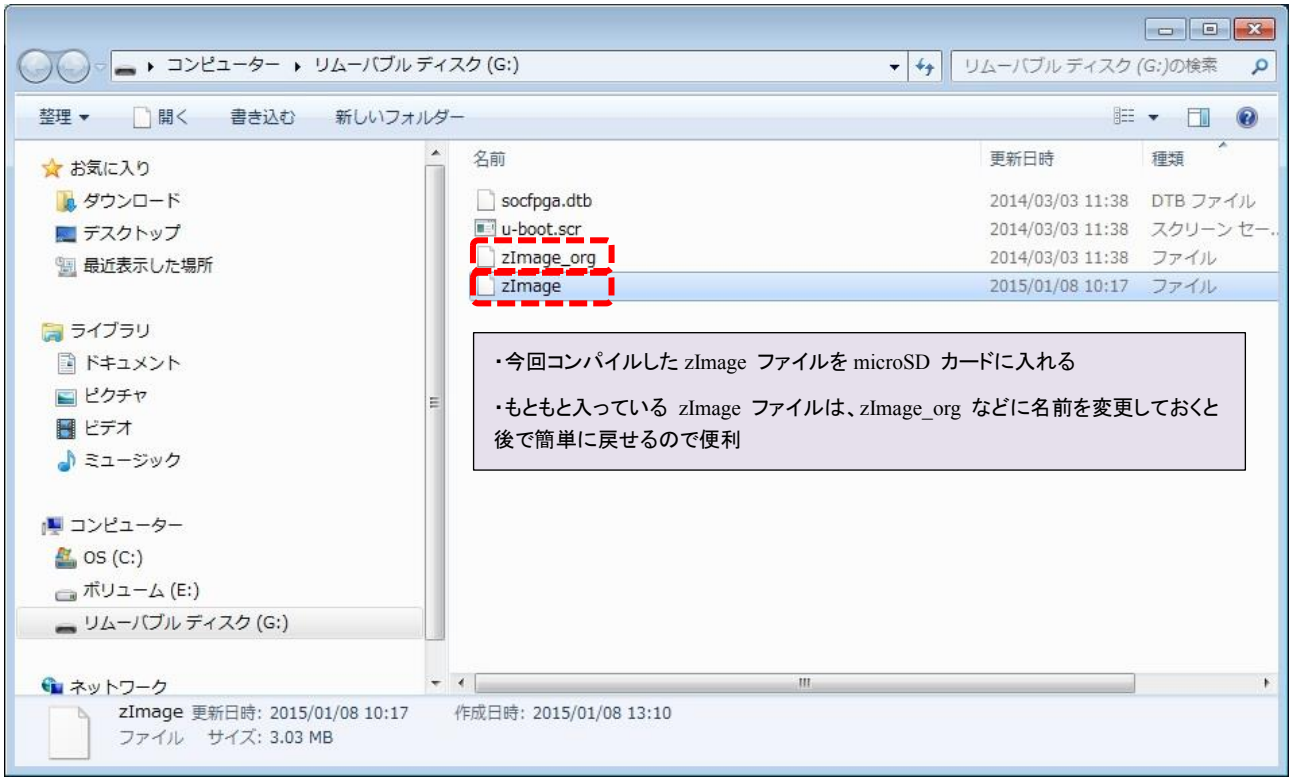
(2) ホスト PC の SD カード・スロット（または USB カード・リーダを接続）に microSD カードを挿します。
下図のような表示が出たら「フォルダを開いてファイルを表示」をクリックします。



【図 2-5-2.1】Windows で microSD カードを開く

(3) カード内の zImage ファイルを、コピーしておいた今回コンパイルしたものに差し換えます。

念のため、もともと入っている zImage ファイルは、 zImage_org などに名前を変更しておくことをお勧めします。



【図 2-5-2.2】microSD カード内の zImage ファイルの入れ換え

(4) zImage ファイルを差し換えたら、ホスト PC から microSD カードを取り出します。

【図 2-5-2.3】microSD カードの取り出し

2-5-3. zImage ファイルを差し換えた microSD カードで Helio を起動する

Helio のカード・スロットに microSD カードを挿して電源を入れます。

Linux カーネル起動時のターミナル表示は次のようになるはずです。

```
Starting kernel ...

Booting Linux on physical CPU 0x0
Initializing cgroup subsys cpuset
Linux version 3.9.0-00161-ged01b8c-dirty (tori@Vine65) (gcc version 4.9.4 (Linaro GCC 4.9-2017.01) ) #1 SMP Tue
Jul 25 11:56:28 JST 2017
```

上記のログから、Linux カーネルのバージョンは 3.9.0 で、gcc のバージョンは 4.9.4、ツール・チェーンは Linaro の GCC を使っていることがわかります。

今回コンパイルした Linux のカーネル・イメージ・ファイルで起動させることができました。

3. カスタム・ドライバの作成とコンパイル（その 1）

カーネルの生成が無事に終了したら、次にカスタム・ドライバを作成しコンパイルして Helio 上で動かしてみましょう。

(1) 第 2 章では、外部のインターネットに接続するため SoC Linux 道場【其ノ弐】で設定した DHCP サーバーを停止させました。この章では再度 Helio と接続するので DHCP サーバーを起動させます。Vine Linux 上で「端末」が起動している場合は、それを一旦終了させてから、再度「端末」を起動することで【其ノ弐】で設定した DHCP サーバーを起動させます。

(2) Linux 上の一般ユーザ（この例では tori）のホーム・ディレクトリ（/home/tori）から以下のコマンドを実行して、Leafpad で【リスト 3.1】に示した内容の hello_driver.c ドライバ・ソース・コードを作成します。

```
[tori@Vine65 ~]$ leafpad hello_driver.c
```

【注記】

「leafpad: ディスプレイをオープンできません:」というエラーが出る場合は、「端末」を一旦終了し再度「端末」を起動してから、上記の leafpad コマンドを再度実行してください。

```
#include <linux/module.h>
#include <linux/init.h>

MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void)
{
 printk(KERN_ALERT "Hello Helio World!!¥n");
 return 0;
}

static void hello_exit(void)
{
 printk(KERN_ALERT "Good-bye Helio World!!¥n");
}

module_init(hello_init);
module_exit(hello_exit);
```

hello_init() 関数の本体。
insmod コマンド実行時に呼び出され、"Hello Helio World!!" のメッセージを出力して、戻り値として 0 を返す

hello_exit() 関数の本体。
rmmod コマンド実行時に呼び出され、" Good-bye Helio World!!" のメッセージを出力

ロード時のエントリポイントの指定は「module_init」マクロで行う。
「module_init」マクロで hello_init() 関数をエントリポイントとして指定。
このエントリポイントは insmod コマンド実行時に呼び出される

アンロード時のエントリポイントの指定は「module_exit」マクロで行う。
「module_exit」マクロで hello_exit() 関数をエントリポイントとして指定。
このエントリポイントは rmmod コマンド実行時に呼び出される

【リスト 3.1】Hello ドライバ・ソース（ファイル名： hello_driver.c）の内容

作成が終了したら、Leafpad で hello_driver.c ファイルを保存して閉じます。

(3) Leafpad で【リスト 3.2】に示した内容の ドライバ用 Makefile を作成します。

```
[tori@Vine65 ~]$ leafpad Makefile
```

注：コマンド行の先頭は、スペースではなく **Tab を入れること！**

```
obj-m := hello_driver.o

all:
	make -C /home/tori/linux-socfpga M=$(PWD) modules

clean:
	make -C /home/tori/linux-socfpga M=$(PWD) clean
```

カーネルを置いたディレクトリを指定する

【リスト 3.2】デバイス・ドライバ用 Makefile の内容

作成が終了したら、Leafpad で Makefile ファイルを保存して閉じます。

【注記】

Makefile の書式として、コマンド行の先頭はスペースではなく **Tab を入力してください**。
スペースではコンパイルがエラーとなるので注意してください。

(4) 以下の **make** コマンドで Hello ドライバをコンパイルします。hello_driver.c と Makefile を作成したディレクトリで実行します（この例では、/home/tori）。

```
[tori@Vine65 ~]$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
make -C /home/tori/linux-socfpga M=/home/tori modules
make[1]: ディレクトリ `/home/tori/linux-socfpga' に入ります
  CC [M]  /home/tori/hello_driver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/tori/hello_driver.mod.o
  LD [M]  /home/tori/hello_driver.ko
make[1]: ディレクトリ `/home/tori/linux-socfpga' から出ます
```

(5) Helio の Linux が起動したら、**Helio（ターゲット）側** で以下のように root でログインして、udhcpc コマンドを実行して、DHCP サーバーから Helio に IP アドレスが付与されるようにします。

【注記】

以降の説明は、既に SoC Linux 道場【其ノ弐】で説明した DHCP サーバーが設定されていて、使用できることを前提としています。

以降の説明では、Linux マシン (Vine Linux) の IP アドレスを 192.168.1.2 として説明しています。IP アドレスがわからない場合は、Vine Linux 上から ifconfig eth0 コマンドを実行することで設定されている IP アドレスを確認することができます。

```
socfpga login: root
root@socfpga:~# udhcpc
udhcpc (v1.20.2) started
Sending discover...
Sending select for 192.168.1.238...
Lease of 192.168.1.238 obtained, lease time 21600
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

(6) 念のため、**Helio 側** から ping コマンドを実行して、ホスト側（Vine Linux）と通信できるか確かめます。

```
root@socfpga:~# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_req=1 ttl=64 time=1.80 ms
64 bytes from 192.168.1.2: icmp_req=2 ttl=64 time=1.35 ms
64 bytes from 192.168.1.2: icmp_req=3 ttl=64 time=1.66 ms
64 bytes from 192.168.1.2: icmp_req=4 ttl=64 time=1.35 ms
      <-------------------------------  「Ctrl」+「c」で終了させます
--- 192.168.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.351/1.544/1.806/0.197 ms
```

(7) Hello ドライバのカーネル・モジュール（hello_driver.ko）が生成されているはずなので、それを NFS 経由で Helio 上に運びます。**ホスト（Vine Linux）側** から以下のコマンドを実行します。

この例では、ターゲット側（Helio）からホスト側（Vine Linux）のマウント先を /opt/Helio にしています。

【注記】

既に SoC Linux 道場【其ノ弐】で説明した、NFS サーバーが設定されていて、使用できることを前提としています。

```
[tori@Vine65 ~]$ cp hello_driver.ko /opt/Helio/
```

【参考】

カーネル・モジュールとは、動的に（カーネルが動作している状態で）カーネルに組み込んで、動作させられるプログラムのことです。一般に拡張子は ".ko" です。

(8) SoC Linux 道場【其ノ四】で説明した、busybox が一般ユーザのホーム・ディレクトリ（この例では、/home/tori）の下にコピーされていることを前提として、以下のコマンドで生成されているファイルを確かめます。

```
[tori@Vine65 ~]$ file busybox
busybox: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter
/lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.31,
BuildID[sha1]=de7760619a2deaccf69611887c141cff67b37dba, stripped
```

(9) FTP サーバーを使って busybox を Helio に運びます。以下のコマンドを **Helio 側** で実行します。

【注記】

以降の説明は、既に SoC Linux 道場【其ノ弐】で説明した FTP サーバーが設定されていて、使用できることを前提としています。

```
root@socfpga:/# cd /tmp
root@socfpga:/tmp# wget ftp://tori:toriumi@192.168.1.2/busybox
--2013-11-05 06:13:29--  ftp://tori:*password*@192.168.1.2/busybox
           => 'busybox'
Connecting to 192.168.1.2:21... connected.
Logging in as tori ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD not needed.
==> SIZE busybox ... 683392
==> PASV ... done.    ==> RETR busybox ... done.
Length: 683392 (667K) (unauthoritative)

100%[======================================>] 683,392      --.-K/s   in 0.04s

2013-11-05 06:13:39 (17.7 MB/s) - 'busybox' saved [683392]
```

(10) **Helio 側** で以下のように、busybox に chmod コマンドで実行権限を与えてから実行します。

```
root@socfpga:/tmp# chmod 755 busybox
root@socfpga:/tmp# ./busybox ls /
bin         home        lost+found  proc        usr
boot        init        media       sbin        var
dev         lib         mnt         sys         www
etc         linuxrc     mount_dir   tmp
```

【注記】

上記の確認で、mount_dir が見当たらない場合は、次の mkdir コマンドで mount_dir ディレクトリを作成してください。

```
root@socfpga:/tmp# mkdir /mount_dir
```

(11) **Helio 側** で以下の busybox のコマンドを実行して、ホスト側（Vine Linux）の /opt/Helio を /mount_dir にマウントします。

```
root@socfpga:/tmp# ./busybox mount -t nfs -o nolock 192.168.1.2:/opt/Helio /mount_dir
```

(12) マウントが完了したら、以下のようにして Hello ドライバのカーネル・モジュール（hello_driver.ko）を実行します。

```
root@socfpga:/tmp# insmod /mount_dir/hello_driver.ko
Hello Helio World!!
root@socfpga:/tmp# lsmod | grep hello_driver
hello_driver            729  0
root@socfpga:/tmp# rmmod hello_driver
Good-bye Helio World!!
```

insmod コマンドで、実行中のカーネルにカーネル・モジュールをロードする。これにより hello_driver 内の hello_init() 関数が呼び出され "Hello Helio World!!" のメッセージが出力される

lsmod コマンドで、ロードされているカーネル・モジュールをリスト表示して、hello_driver がロードされているかを確認する

rmmod コマンドで、実行中のカーネルからカーネル・モジュールをアンロードする。これにより hello_driver 内の hello_exit () 関数が呼び出され " Good-bye Helio World!!" のメッセージが出力される

【注記】

Helio 上で動作している microSD カード内の Linux と、Vine Linux（ホスト PC）上でビルドした Linux ソースのバージョンが異なると、上記の insmod コマンドを実行したときに、下記のようなエラーが出ます。

```
hello_driver: version magic '3.9.0-00161-ged01b8c SMP mod_unload ARMv7 p2v8 ' should be '3.9.0 SMP mod_unload ARMv7 p2v8 '
Error: could not insert module /mount_dir/hello_driver.ko: Invalid module format
```

このような場合は、SD カード内の Linux カーネル（zImage）を、Vine Linux（ホスト PC）上でコンパイルして生成した zImage と差し替えてから再度 insmod コマンドを試してください。

これでカスタム・ドライバの作成に必要な環境が整い、カスタム・ドライバをターゲット上で実行できることを確認できました。

次回の SoC Linux 道場【其ノ六】「カスタム・ドライバの作成とコンパイル（その 2）」では、Helio 用リファレンス・デザインを入手して Helio にダウンロードします。

そして、このリファレンス・デザイン内に含まれる LED を制御するために、カスタム・ドライバを記述してコンパイルします。

作成した LED 制御カスタム・ドライバを Helio に NFS 経由で転送して実行し、ドライバによる LED アクセスの状態を確認します。

改版履歴

Revision	年月	概要
1	2015 年 2 月	新規作成
2	2017 年 8 月	① Sourcery CodeBench Lite Edition for ARM GNU/Linux の配布終了に伴い、クロス・コンパイル環境として Linaro Toolchain を使用する説明に変更 ② ゲスト OS を Vine Linux 6.2.1 i686 から Vine Linux 6.5 x86_64 に変更